```
##################################Boot#########################################
# Complete project details at https://RandomNerdTutorials.com

import time
from umqttsimple import MQTTClient
import ubinascii
import machine
import micropython
import network
import esp
esp.osdebug(None)
import gc
gc.collect()

ssid = 'VodafoneMobileWiFi-3BCC3D'
password = 'rnPa719563'
mqtt_server = '192.168.0.123'
#EXAMPLE IP ADDRESS
#mqtt_server = '192.168.1.144'
client_id = ubinascii.hexlify(machine.unique_id())
topic_sub = 'teste/led'
topic_pub = 'teste/led'

last_message = 0
message_interval = 0.5
counter = 0

station = network.WLAN(network.STA_IF)

station.active(True)
station.connect(ssid, password)

while station.isconnected() == False:
  pass

print('Connection successful')
print(station.ifconfig())


##############################main#############################################################

Complete project details at https://RandomNerdTutorials.com

#
from machine import Pin, ADC
from time import sleep
#

def sub_cb(topic, msg):
  print((topic, msg))
  print ('xxxx')
  if topic == b'notification' and msg == b'received':
    print('ESP received hello message')

def connect_and_subscribe():
```

```
  global client_id, mqtt_server, topic_sub
  client_id = 'xxx222'
  client = MQTTClient(client_id, mqtt_server)
  # client = MQTTClient(CLIENT_ID, MQTT_SERVERI MQTT_PORT, MQTT_USER, MQTT_PASSWORD)
  client.set_callback(sub_cb)
  client.connect()
  client.subscribe(topic_sub)
  print('Connected to %s MQTT broker, subscribed to %s topic' % (mqtt_server, topic_sub))
  return client

def restart_and_reconnect():
  print('Failed to connect to MQTT broker. Reconnecting...')
  time.sleep(10)
  machine.reset()

try:
  client = connect_and_subscribe()
except OSError as e:
  restart_and_reconnect()


# Config Entrada analógca
pot = machine.ADC(machine.Pin(35))
pot.atten (machine.ADC.ATTN_11DB)       # full range 3.3V
pot.width (machine.ADC.WIDTH_12BIT)     # 0..4096

#
v = 3.3
v = v/1000

a0 = 0.2265846
a1 = 24152.109
a2 = 67233.425
a3 = 2210340.7
a4 = -860963915
a5 = 4.83506*(10**10)
a6 = -1.18452*(10**12)
a7 = 1.38690*(10**13)
a8 = -6.33708*(10**13)

gain = 1700
cjc = 20

while True:
    sleep(0.5)
    xx = pot.read()
    v = xx*0.000805 /gain
    T = a0 + a1 * v + a2 * (v**2) +a3 * (v**3) + a4 * (v**4) +cjc
#

# Resolução 3.3 / 4096 = 0.000805
#aux = 1
#while aux<100:
#  time.sleep (0.1)
#  xx = pot.read()
```

```
  #print ( 'Valor ADC : ', xx,'V in : ', xx * 0.000805 )
#  vin = xx * 0.000805

    try:
        client.check_msg()
        if (time.time() - last_message) > message_interval:

            msg = b'Hello #%d' % counter
            topic_pub2 = 'teste/led'
            client.publish(topic_pub2, msg)

            msg = str (T) #'Vin = {:0.2f} ',format(vin)  #= %f0.2', vin #vin em vez de T
            topic_pub1 = 'teste/nivel'
            client.publish(topic_pub1, msg)


            last_message = time.time()
            counter += 1
    except OSError as e:
        restart_and_reconnect()


#######################umqtt.simple#############################################################
try:
    import usocket as socket
except:
    import socket
import ustruct as struct
from ubinascii import hexlify

class MQTTException(Exception):
    pass

class MQTTClient:

    def __init__(self, client_id, server, port=0, user=None, password=None, keepalive=0,
                 ssl=False, ssl_params={}):
        if port == 0:
            port = 8883 if ssl else 1883
        print ('Port : ', port )
        print ('CLient : ', client_id )
        print ('Server :' , server )
        print ('user : ',user )

        self.client_id = client_id
        self.sock = None
        self.server = server
        self.port = port
        self.ssl = ssl
        self.ssl_params = ssl_params
        self.pid = 0
        self.cb = None
        self.user = user
        self.pswd = password
```

```
    self.keepalive = keepalive
    self.lw_topic = None
    self.lw_msg = None
    self.lw_qos = 0
    self.lw_retain = False

def _send_str(self, s):
    self.sock.write(struct.pack("!H", len(s)))
    self.sock.write(s)

def _recv_len(self):
    n = 0
    sh = 0
    while 1:
        b = self.sock.read(1)[0]
        n |= (b & 0x7f) << sh
        if not b & 0x80:
            return n
        sh += 7

def set_callback(self, f):
    self.cb = f

def set_last_will(self, topic, msg, retain=False, qos=0):
    assert 0 <= qos <= 2
    assert topic
    self.lw_topic = topic
    self.lw_msg = msg
    self.lw_qos = qos
    self.lw_retain = retain

def connect(self, clean_session=True):
    self.sock = socket.socket()
    addr = socket.getaddrinfo(self.server, self.port)[0][-1]
    self.sock.connect(addr)
    if self.ssl:
        import ussl
        self.sock = ussl.wrap_socket(self.sock, **self.ssl_params)
    premsg = bytearray(b"\x10\0\0\0\0\0")
    msg = bytearray(b"\x04MQTT\x04\x02\0\0")

    sz = 10 + 2 + len(self.client_id)
    msg[6] = clean_session << 1
    if self.user is not None:
        sz += 2 + len(self.user) + 2 + len(self.pswd)
        msg[6] |= 0xC0
    if self.keepalive:
        assert self.keepalive < 65536
        msg[7] |= self.keepalive >> 8
        msg[8] |= self.keepalive & 0x00FF
    if self.lw_topic:
        sz += 2 + len(self.lw_topic) + 2 + len(self.lw_msg)
        msg[6] |= 0x4 | (self.lw_qos & 0x1) << 3 | (self.lw_qos & 0x2) << 3
        msg[6] |= self.lw_retain << 5
```

```
    i = 1
    while sz > 0x7f:
        premsg[i] = (sz & 0x7f) | 0x80
        sz >>= 7
        i += 1
    premsg[i] = sz

    self.sock.write(premsg, i + 2)
    self.sock.write(msg)
    #print(hex(len(msg)), hexlify(msg, ":"))
    self._send_str(self.client_id)
    if self.lw_topic:
        self._send_str(self.lw_topic)
        self._send_str(self.lw_msg)
    if self.user is not None:
        self._send_str(self.user)
        self._send_str(self.pswd)
    resp = self.sock.read(4)
    assert resp[0] == 0x20 and resp[1] == 0x02
    if resp[3] != 0:
        raise MQTTException(resp[3])
    return resp[2] & 1

def disconnect(self):
    self.sock.write(b"\xe0\0")
    self.sock.close()

def ping(self):
    self.sock.write(b"\xc0\0")

def publish(self, topic, msg, retain=False, qos=0):
    print ( ' **** Publish ....' )
    print ( 'topic : ', topic)
    print ( 'Msg : ', msg )
    pkt = bytearray(b"\x30\0\0\0")
    pkt[0] |= qos << 1 | retain
    sz = 2 + len(topic) + len(msg)
    if qos > 0:
        sz += 2
    assert sz < 2097152
    i = 1
    while sz > 0x7f:
        pkt[i] = (sz & 0x7f) | 0x80
        sz >>= 7
        i += 1
    pkt[i] = sz
    #print(hex(len(pkt)), hexlify(pkt, ":"))
    self.sock.write(pkt, i + 1)
    self._send_str(topic)
    if qos > 0:
        self.pid += 1
        pid = self.pid
        struct.pack_into("!H", pkt, 0, pid)
        self.sock.write(pkt, 2)
    self.sock.write(msg)
```

```
    if qos == 1:
        while 1:
            op = self.wait_msg()
            if op == 0x40:
                sz = self.sock.read(1)
                assert sz == b"\x02"
                rcv_pid = self.sock.read(2)
                rcv_pid = rcv_pid[0] << 8 | rcv_pid[1]
                if pid == rcv_pid:
                    return
    elif qos == 2:
        assert 0

def subscribe(self, topic, qos=0):
    assert self.cb is not None, "Subscribe callback is not set"
    pkt = bytearray(b"\x82\0\0\0")
    self.pid += 1
    struct.pack_into("!BH", pkt, 1, 2 + 2 + len(topic) + 1, self.pid)
    #print(hex(len(pkt)), hexlify(pkt, ":"))
    self.sock.write(pkt)
    self._send_str(topic)
    self.sock.write(qos.to_bytes(1, "little"))
    while 1:
        op = self.wait_msg()
        if op == 0x90:
            resp = self.sock.read(4)
            #print(resp)
            assert resp[1] == pkt[2] and resp[2] == pkt[3]
            if resp[3] == 0x80:
                raise MQTTException(resp[3])
            return

# Wait for a single incoming MQTT message and process it.
# Subscribed messages are delivered to a callback previously
# set by .set_callback() method. Other (internal) MQTT
# messages processed internally.
def wait_msg(self):
    res = self.sock.read(1)
    self.sock.setblocking(True)
    if res is None:
        return None
    if res == b"":
        raise OSError(-1)
    if res == b"\xd0":  # PINGRESP
        sz = self.sock.read(1)[0]
        assert sz == 0
        return None
    op = res[0]
    if op & 0xf0 != 0x30:
        return op
    sz = self._recv_len()
    topic_len = self.sock.read(2)
    topic_len = (topic_len[0] << 8) | topic_len[1]
    topic = self.sock.read(topic_len)
    sz -= topic_len + 2
```

```
        if op & 6:
            pid = self.sock.read(2)
            pid = pid[0] << 8 | pid[1]
            sz -= 2
        msg = self.sock.read(sz)
        self.cb(topic, msg)
        if op & 6 == 2:
            pkt = bytearray(b"\x40\x02\0\0")
            struct.pack_into("!H", pkt, 2, pid)
            self.sock.write(pkt)
        elif op & 6 == 4:
            assert 0

    # Checks whether a pending message from server is available.
    # If not, returns immediately with None. Otherwise, does
    # the same processing as wait_msg.
    def check_msg(self):
        self.sock.setblocking(False)
        return self.wait_msg()


################################temperatura#################################################

from machine import Pin, ADC
from time import sleep

aux = True

pot = ADC(Pin(35))
pot.atten(ADC.ATTN_11DB)
pot.width(ADC.WIDTH_12BIT)

v = 3.3
v = v/1000

a0 = 0.2265846
a1 = 24152.109
a2 = 67233.425
a3 = 2210340.7
a4 = -860963915
a5 = 4.83506*(10**10)
a6 = -1.18452*(10**12)
a7 = 1.38690*(10**13)
a8 = -6.33708*(10**13)

gain = 1700
cjc = 20

while True:
    sleep(0.5)
    xx = pot.read()
    v = xx*0.000805 /gain
    T = a0 + a1 * v + a2 * (v**2) +a3 * (v**3) + a4 * (v**4) +cjc
    print ( 'aux: ', aux, 'Valor ADC :', xx, 'V in : ',v, ' - T em ºC: - ', T)
```

```
########################caudal#############################################################

from machine import Pin,ADC
from time import sleep
import time


aux=True
ciclo=1
a1=0
a2=0
a3=0
a4=0
pulso=0
periodo=0
freq=0
t1=0.05 #intervalo de tempo de cada leitura de a1, a2
t2=2 #intervalo de tempo de leitura
t_end = time.time() + t2


meiospulsos=0
while time.time() < t_end:
    sleep(t1)
    aux=not aux
    valorADC=ADC(Pin(33))
    adc=valorADC.read()
    adc01=adc/4095
    if ciclo==1 and adc01==1:
        a1=a1+1
    elif adc01==0 and a1!=0:
        a2=a2+1
        ciclo=0
    elif  adc01==1 and a2!=0:
        ciclo=1
        a1=0
        a2=0
        meiospulsos=meiospulsos+1
        periodo=t2/meiospulsos
        freq=1/(periodo)
    print ('a1:',a1, 'a2:',a2, 'periodo:',periodo, 'frequencia', freq, 'meios pulsos:', meiospulsos)
```